Paralelización Parcial del Algoritmo de CANNY en CUDA C++

Jhonatan Barrera, *Estudiante, Ingeniería de Sistemas y Computación,*
Angie Jaramillo, *Estudiante, Ingeniería de Sistemas y Computación*

Resumen—En este documento se presenta una comparativa de tiempos de ejecución de la implementación parcial del Detector de Bordes de CANNY en su versión secuencial sobre CPU y dos implementaciones que hacen uso de técnicas de paralelización sobre GPU mediante el uso de CUDA. Se mostrará entonces cómo a partir de estas técnicas de computación en paralelo se obtiene una reducción considerable del tiempo de ejecución dada una aceleración que aumenta a medida de que las dimensiones de la imagen a procesar son mayores.

Palabras Clave—CUDA, Parallelization, Canny Edge Detector, Non-Maximum Suppression.

1. Introducción

El propósito de la detección de bordes en general es reducir significativamente la cantidad de datos en una imagen, siempre y cuando se preserven las propiedades estructurales que se utilizan en el procesamiento de imágenes. Para este fin existen varios algoritmos, y para el propósito de este trabajo estudiaremos el algoritmo desarrollado por John F. Canny (JFC) en 1986 [1].

Este algoritmo a pesar de ser antiguo se ha convertido en uno de los métodos de detección de bordes estándar y sigue siendo usado en la investigación.

1.1. Criterios de CANNY

JFC desarrollo un algoritmo que cumpliera los siguientes tres criterios:

1. Detección:
   El algoritmo debe detectar el mayor numero de bordes reales. Esto corresponde a maximizar la relación de señal-ruido.
2. Localización:
   Los bordes detectados deben estar lo más cerca posible a los bordes reales.
3. Respuesta mínima:
   El borde de una imagen solo debe ser detectado una vez, y siempre que sea posible el ruido de la imagen no debe crear falsos bordes.

2. Algoritmo de Detección de Bordes de CANNY

Este algoritmo se encuentra separado en cuatro pasos:

1. Suavizado: Desenfoque de la imagen para eliminar el ruido.
2. Determinación de Gradientes: Los bordes se marcan en donde la magnitud de los gradientes es mayor.
3. Supresión No Máxima: Solo los locales máximos deben marcarse como bordes.
4. Doble umbralización: Los potenciales bordes deben estar determinados por umbralización.
5. Seguimiento por histéresis:Los bordes finales se determinan la supresión de todos los bordes que no están conectados a un borde determinado (fuerte).

Sin entrar en detalles, a continuación presentamos una ampliación de los pasos implementados.

2.1. Suavizado

Todas las imágenes tomadas desde una cámara contienen cierta cantidad de ruido. Este ruido se puede confundir con los bordes, el primer paso entonces consiste en suavizar la imagen para aislarla del ruido, en este caso se aplico un filtro de Gauss a la imagen.

Kernel usado en el filtro. ($\sigma = 1{,}4$)

$$Kgauss = 1/159 \begin{pmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{pmatrix}$$

2.2. Determinación de Gradientes

Los bordes encontrados por Canny pertenecen a los pixeles donde la intensidad de la escala de grises cambia. Estos pixeles se obtienen mediante la determinación de los gradientes de la imagen en 'x' y 'y' a través del operador de Sobel.

$$Gx = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

• *Agradecemos al Ingeniero John Osorio por sus aportes y recomendaciones a lo largo de este semestre.*

$$Gy = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

A partir de estos gradientes se calculan los puntos fuertes de borde mediante la formula (2.1)

$$|G| = \sqrt{Gx^2 + Gy^2} \tag{2.1}$$

También se debe calcular el valor de theta que será usado en la Supresión No Máxima:

$$\theta = \arctan \frac{|Gx|}{|Gy|} \tag{2.2}$$

2.3. Supresión No Máxima

Este paso busca encontrar los bordes débiles que son paralelos a los bordes fuertes y eliminarlos. El algoritmo para cada pixel de la imagen de gradiente es:

1. Al rededor de la dirección del gradiente θ mas cercano a 45°
2. Comparar la resistencia del borde de píxel actual con la resistencia de los bordes en dirección del píxel en el gradiente positivo y negativo. Es decir, si la dirección del gradiente es norte (θ = 90°), comparar con los pixeles hacia el norte y el sur.
3. Si la resistencia del borde del píxel actual es el más grande; preservar el valor de la resistencia de los bordes. Si no es así, suprimir el valor.

3. Trabajo Realizado

En este trabajo se implementó parcialmente el algoritmo de Canny en el lenguaje de programación C++, esta implementación la llamamos Secuencial. Se realizaron también dos implementaciones que hacen uso de técnicas de paralelización (sobre CUDA C++), como lo son la paralelización 'base' que hace uso de Memoria Global, y un uso especifico para valores que no cambian a lo largo de la ejecución, Memoria Constante, de manera que se declaran los kernel de Gauss y Sobel como constantes.

El objetivo de estas tres implementaciones es analizar sus tiempos de ejecución y obtener la aceleración resultante de este proceso. Se espera que las implementaciones en paralelo optimicen la ejecución del algoritmo en cuanto a tiempo de ejecución se refiere, mostrando de esta manera que la paralelización es una gran herramienta en la detección de bordes de imágenes complejas al reducir el tiempo de detección de los mismos.

4. Análisis y Resultados

Para realizar el análisis de estos algoritmos se considero un DataSet de imágenes con tamaños matriciales desde 580x580 hasta 19843x8504. Se tomaron 20 muestras para cada una de estas imágenes, de los cuales se obtiene los promedios de tiempo a continuación:

Imagen	Secuencial	Global	Constante
580X580	0,0687071	0,00281205	0,0023816
638X640	0,08357675	0,00334405	0,0028545
1366X768	0,21104265	0,0081652	0,0068586
2560X1600	0,82379125	0,02753685	0,02264345
4928X3264	3,4296161	0,10075015	0,08447555
5226X4222	4,62425675	0,1385763	0,11664495
12000X6000	14,5266978	0,44735335	0,3770079
12000X9000	21,528332	0,67437875	0,56628045
19843X8504	33,50886415	1,052265	0,8840947

Cuadro 1
Tiempos de ejecución para los algoritmos Secuencial, y Paralelo (Global, Constante).

Figura 1. Tiempo de ejecución para el algoritmo secuencial.

Como se observa en el Cuadro 1, a medida que la matriz de entrada aumenta el tiempo de ejecución también lo hace, sin embargo esto apenas y se percibe para las implementaciones en paralelo.

En las Figuras 2 y 3, se confirma que el tiempo de ejecución para las implementaciones en paralelo es notoria, obteniéndose una aceleración muy cercana para las versiones en paralelo.

5. Conclusiones

- Se verificó la factibilidad de paralelizar parcialmente el algoritmo de detección de bordes de Canny, obteniendo resultados favorables en la aceleración

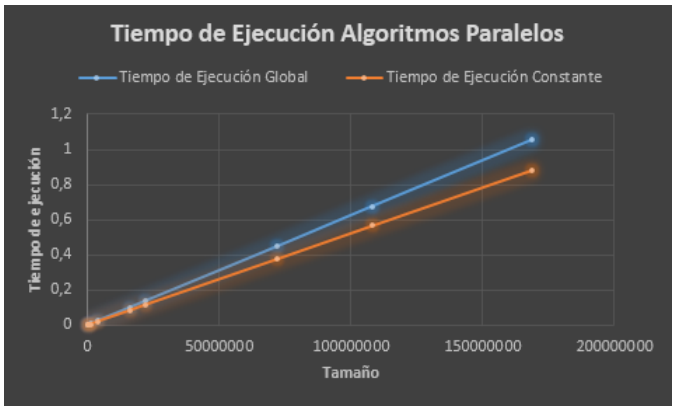

Figura 2. Tiempo de ejecución algoritmos paralelos.

Imagen	Sec. vs Glbl	Sec vs Cons	Glbl vs Cons
580X580	24,43310041	28,84913504	1,180739839
638X640	24,99267355	29,27894552	1,171501139
1366X768	25,84659898	30,77051439	1,190505351
2560X1600	29,91595807	36,38099539	1,216106645
4928X3264	34,04080391	40,59892004	1,192654561
5226X4222	33,3697519	39,64386585	1,188017998
12000X6000	32,47253608	38,53154748	1,186588796
12000X9000	31,92320636	38,017085	1,19089181
19843X8504	31,84451079	37,90189462	1,190217519

Cuadro 2
Aceleración obtenida con el uso de los algoritmos paralelos respecto al secuencial, la última columna representa la aceleración respecto a los algoritmos.

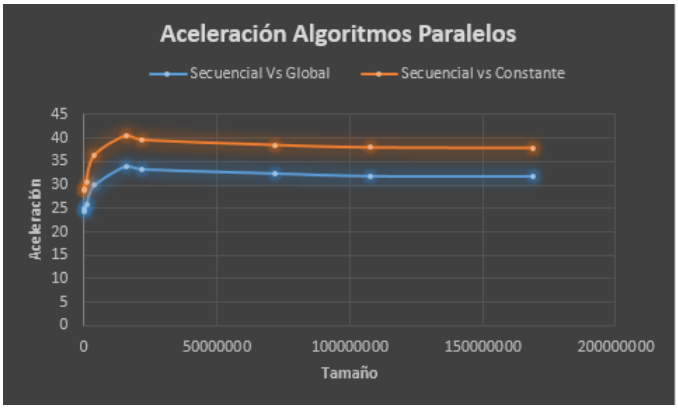

Figura 3. Aceleración obtenida a partir de las implementaciones en CUDA.

del algoritmo.

- Para las dos implementaciones en paralelo que se realizaron se obtuvo una aceleración que permite bajar los tiempos de ejecución entre un $96,9 - 97,4\,\%$ para la prueba con una imagen de matriz 19843x8504.

- La diferencia en la aceleración que se obtuvo entre las versiones paralelas implementadas difiere por poco sin embargo el uso de la memoria constante mejora la implementación sobre memoria global.

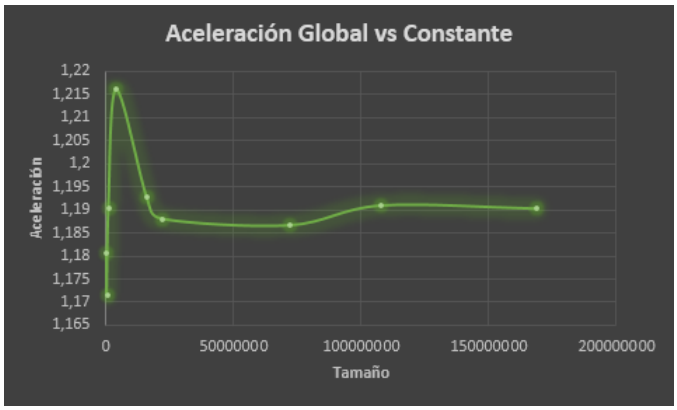

Figura 4. Aceleración entre las dos implementaciones en paralelo.

- En el análisis de la aceleración obtenida a partir de estas implementaciones paralelas se puede observar que este valor tiende a un valor constate.

- Se debe recordar que la maquina usada para la toma de los datos no es de uso específico y esto puede introducir ruido en los tiempos de ejecución.

REFERENCIAS

[1] John Canny. A computational approach to edge detection. Pattern Analysis and Machine Intelligence, IEEE Transactions on , PAMI-8(6):679–698, Nov. 1986